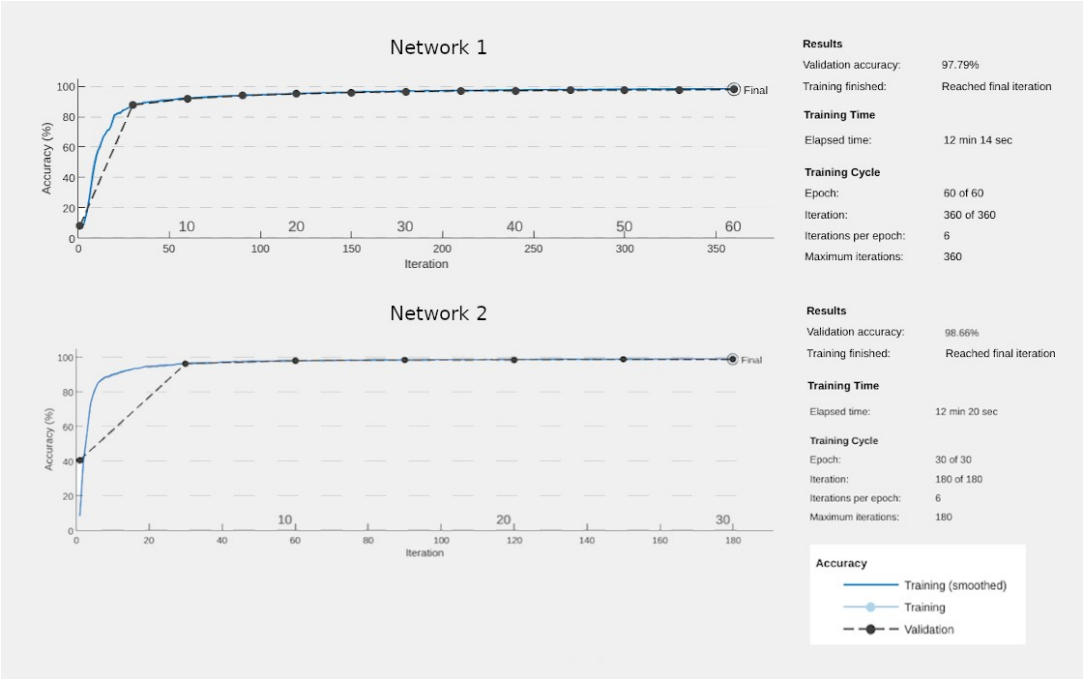
Homework 3

Problem 1: Convolutional networks

Following the instructions given we construct two networks and use data from the MNIST database to train them. The training progress for both networks is show in the following figure:



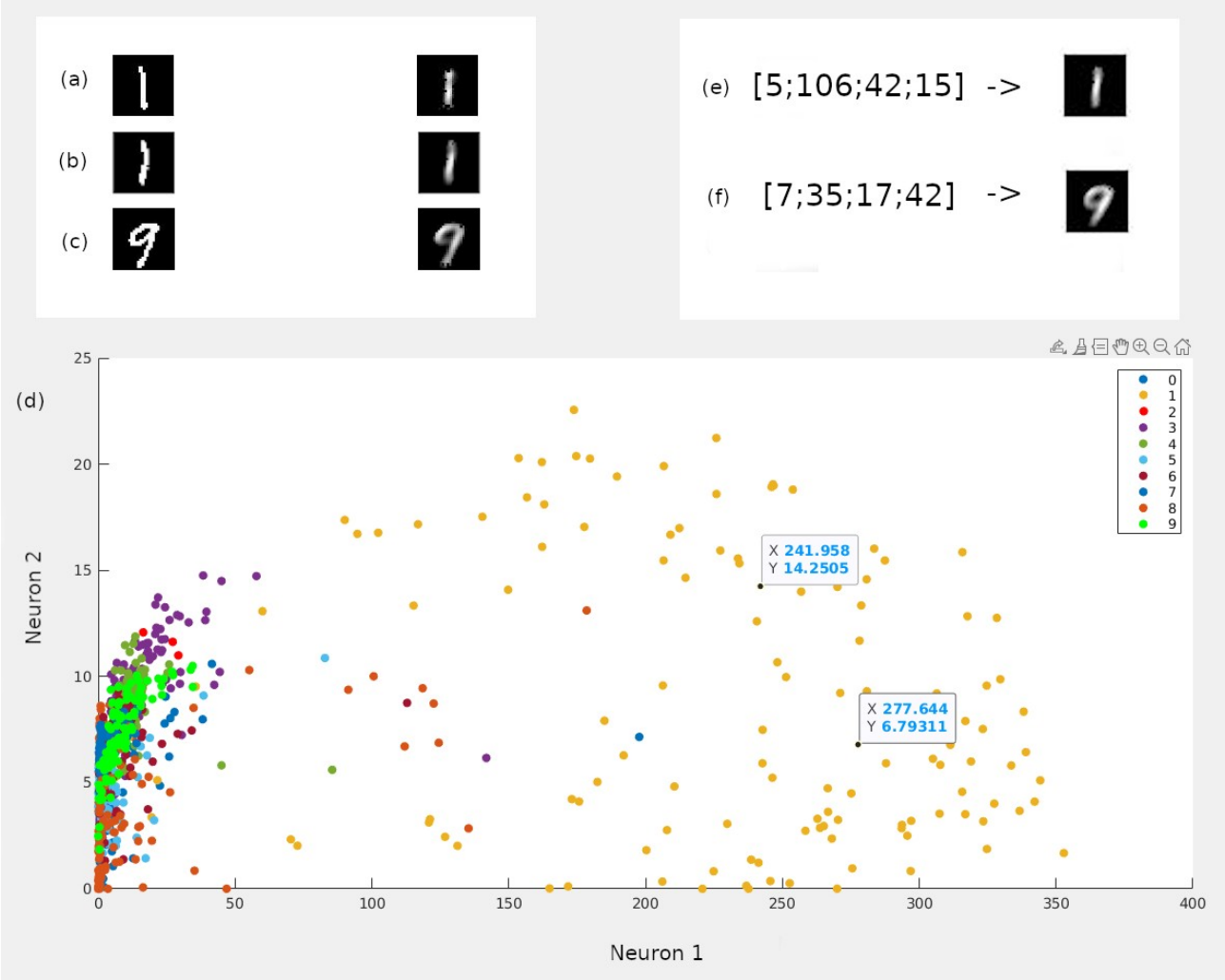
We can see that the second network reaches higher validation accuracy faster than the first network and the overall validation accuracy of the second network is better. We notice the same when we calculate the errors obtained on the training, validation and test sets:

	Network 1	**Network 2**
Training Set Error	0.0162	0.0094
Validation Set Error	0.0221	0.0134
Test Set Error	0.0192	0.0120

The second network performs better than the first because it is deeper and this way the multiple layers can learn more intermediate features between the raw data and the high-level classification. For example the first layer will train itself to recognize very basic things like edges while the next layer will recognise collection of edges such as shapes. Thus deeper layers will learn more complicated and higher-order features.

Problem 2: Fully connected Autoencoder

Following the instructions given we construct two autoenconders and use data from the MNIST database to train them. After 2400 epochs the first network can reproduce convincingly the digit ‘1’ while the second network the digits ‘1’ and ‘9’. We compare the input and output of the networks as a montage in (a) for autoencoder 1 and in (b) and (c) for autoencoder 2.



We divide each trained network into an encoder and a decoder. For the autoencoder 1 we made the scatter plot (d) of the bottleneck neurons using the testing dataset. The yellow color is used for digit ‘1’ which is the one well reproduced. We notice that any value of neuron 1 above 200 corresponds to digit 1. To test this rule we choose two random points that follow the rule and feed them to the decoder. Our points are (241.958,14.2505), (277.644,6.79311) and when used as inputs to the decoder it reproduces the digit ‘1’. Then in order to find the coding rule for the well-classified digits for autoencoder 2, we find 16 handwritten digits from the testing set, 8 for each well-reproduced digit. We notice that the values for the four neurons are very similar so we calculate the average values of neuron for each digit. For digit ‘1’ we get the values [5;106;42;15] and digit ‘9’ the values [7;35;17;42]. Thus when we feed them to the decoder 2 it reproduces the expected digits as shown in (e) and (f). The same rule doesn’t apply for o digits where the valuesbetween neurons of the same input are similar and don’t have big difference like for example the average value of neuron 1 for digit ‘1’ which is 5 and the value of neuron 2 for the same digit which is 106.

Appendix A

For the first problem I wrote the following programm in MATLAB:

```
clear all;

exerciseNumber = 4;
[xTrain, tTrain, xValid, tValid, xTest, tTest] = LoadMNIST(exerciseNumber);

options = trainingOptions('sgdm', ...
  'Momentum',0.9, ...
  'InitialLearnRate',0.001, ...
  'MaxEpochs',60, ...
  'MiniBatchSize',8192, ...
  'Shuffle', 'every-epoch', ...
  'ValidationData',{xValid, tValid}, ...
  'ValidationFrequency',30, ...
  'ValidationPatience',5, ...
  'Plots','training-progress');

layers = [
  imageInputLayer([28 28 1]);
  convolution2dLayer(5, 20, 'Padding', 1, 'WeightsInitializer', 'narrow-normal');
  reluLayer
  maxPooling2dLayer(2, 'Stride', 2);
  fullyConnectedLayer(100, 'WeightsInitializer', 'narrow-normal');
  reluLayer
  fullyConnectedLayer(10, 'WeightsInitializer', 'narrow-normal');
  softmaxLayer
  classificationLayer
];

network1 = trainNetwork(xTrain, tTrain, layers, options);

trainPred1 = classify(network1, xTrain);
validPred1 = classify(network1, xValid);
testPred1 = classify(network1, xTest);

trainError1 = sum(trainPred1 ~= tTrain)./numel(tTrain);
validError1 = sum(validPred1 ~= tValid)./numel(tValid);
testError1 = sum(testPred1 ~= tTest)./numel(tTest);

fprintf('The training set error is: %.4f\n.', trainError1);
fprintf('The validation set error is: %.4f\n.', validError1);
fprintf('The test set error is: %.4f\n.', testError1);

clear all;

exerciseNumber = 4;
[xTrain, tTrain, xValid, tValid, xTest, tTest] = LoadMNIST(exerciseNumber);
```

```
options = trainingOptions('sgdm', ...
  'Momentum',0.9, ...
  'InitialLearnRate',0.01, ...
  'MaxEpochs',30, ...
  'MiniBatchSize',8192, ...
  'Shuffle', 'every-epoch', ...
  'ValidationData',{xValid, tValid}, ...
  'ValidationFrequency',30, ...
  'ValidationPatience',5, ...
  'Plots','training-progress');

layers = [
  imageInputLayer([28 28 1]);
  convolution2dLayer(3, 20, 'Padding', 1, 'WeightsInitializer', 'narrow-normal');
  batchNormalizationLayer
  reluLayer
  maxPooling2dLayer(2, 'Stride', 2);
  convolution2dLayer(3, 30, 'Padding', 1, 'WeightsInitializer', 'narrow-normal');
  batchNormalizationLayer
  reluLayer
  maxPooling2dLayer(2, 'Stride', 2);
  convolution2dLayer(3, 50, 'Padding', 1, 'WeightsInitializer', 'narrow-normal');
  batchNormalizationLayer
  reluLayer
  fullyConnectedLayer(10, 'WeightsInitializer', 'narrow-normal');
  softmaxLayer
  classificationLayer
];

network2 = trainNetwork(xTrain, tTrain, layers, options);

trainPred2 = classify(network2, xTrain);
validPred2 = classify(network2, xValid);
testPred2 = classify(network2, xTest);

trainError2 = sum(trainPred2 ~= tTrain)./numel(tTrain);
validError2 = sum(validPred2 ~= tValid)./numel(tValid);
testError2 = sum(testPred2 ~= tTest)./numel(tTest);

fprintf('The training set error is: %.4f\n.', trainError1);
fprintf('The validation set error is: %.4f\n.', validError1);
fprintf('The test set error is: %.4f\n.', testError1);
```

Appendix B

For the second problem I wrote the following programm in MATLAB:

```
% Network 1


clear all;

exerciseNumber = 4;
[xTrain, tTrain, xValid, tValid, xTest, tTest] = LoadMNIST(exerciseNumber);
numOfImages = size(xTrain, 4);
xTrain = reshape(xTrain, 784, numOfImages)./255;
xValid = reshape(xValid, 784, size(xValid,4))./255;

options = trainingOptions('adam', ...
  'InitialLearnRate',0.001, ...
  'MaxEpochs',800, ...
  'MiniBatchSize',8192, ...
  'Shuffle', 'every-epoch', ...
  'ValidationData',{xValid, xValid}, ...
  'ValidationFrequency',30, ...
  'ValidationPatience',5, ...
  'Plots','training-progress', ...
  'ExecutionEnvironment', 'gpu');

layers = [
  sequenceInputLayer(784);
  fullyConnectedLayer(50, 'WeightsInitializer', 'glorot');
  reluLayer
  fullyConnectedLayer(2, 'WeightsInitializer', 'glorot');
  reluLayer
  fullyConnectedLayer(784, 'WeightsInitializer', 'glorot');
  reluLayer
  regressionLayer
];

network1 = trainNetwork(xTrain, xTrain, network1.Layers, options);
trainPred1 = predict(network1, xTrain);

layers_encode1 = network1.Layers(1:5);
layers_encode1(6) = regressionLayer;
encoder1 = assembleNetwork(layers_encode1);

layers_decode1 = sequenceInputLayer(2);
layers_decode1(2:4) = network1.Layers(6:8);
decoder1 = assembleNetwork(layers_decode1);

testDataSize = size(xTest, 4);
xTest1 = reshape(xTest, 784, testDataSize)./255;
testEnc1 = predict(encoder1, xTest1);
testDec1 = predict(decoder1, testEnc1);

save('Net1', 'network1'); % save network to resume training later

clear all;

% Network 2

exerciseNumber = 4;
[xTrain, tTrain, xValid, tValid, xTest, tTest] = LoadMNIST(exerciseNumber);
```

```
numOfImages = size(xTrain, 4);
xTrain = reshape(xTrain, 784, numOfImages)./255;
xValid = reshape(xValid, 784, size(xValid,4))./255;

options = trainingOptions('adam', ...
  'InitialLearnRate',0.001, ...
  'MaxEpochs',800, ...
  'MiniBatchSize',8192, ...
  'Shuffle', 'every-epoch', ...
  'ValidationData',{xValid, xValid}, ...
  'ValidationFrequency',30, ...
  'ValidationPatience',5, ...
  'Plots','training-progress', ...
  'ExecutionEnvironment', 'gpu');

layers = [
  sequenceInputLayer(784);
  fullyConnectedLayer(50, 'WeightsInitializer', 'glorot');
  reluLayer
  fullyConnectedLayer(4, 'WeightsInitializer', 'glorot');
  reluLayer
  fullyConnectedLayer(784, 'WeightsInitializer', 'glorot');
  reluLayer
  regressionLayer
];

network2 = trainNetwork(xTrain, xTrain, network2.Layers, options);
trainPred2 = predict(network2, xTrain);

layers_encode2 = network2.Layers(1:5);
layers_encode2(6) = network2.Layers(8);
encoder2 = assembleNetwork(layers_encode2);

layers_decode2 = sequenceInputLayer(4);
layers_decode2(2:4) = network2.Layers(6:8);
decoder2 = assembleNetwork(layers_decode2);

testDataSize = size(xTest, 4);
xTest2 = reshape(xTest, 784, testDataSize)./255;
testEnc2 = predict(encoder2, xTest2);
testDec2 = predict(decoder2, testEnc2);

save('Net2', 'network2'); % save network to resume training later
```